INFORMATION AND COMPUTATION 97, 234–261 (1992)

The Synchronization of Nonuniform Networks of Finite Automata

TAO JIANG*

Department of Computer Science and Systems, McMaster University, Hamilton, Ontario, Canada L8S 4K1

The generalized firing squad synchronization problem (gfssp) is the well-known firing squad synchronization problem (fssp) extended to arbitrarily connected networks of finite automata. Here, the transmission delays associated with the links of a network are assumed to be 0; i.e., a signal can get through a link in no time. When the delays are allowed to be arbitrary nonnegative integers, the problem is called gfssp-nud (i.e., gfssp with nonuniform delays). We give for the first time a solution of gfssp-nud. The solution is independent of the structure of the network and the actual delays of the links. The firing time of the solution is bounded by $O(\Delta^3 + \tau_{\max})$, where $\tau_{\max}$ is the maximum transmission delay of any single link and Δ is the maximum transmission delay between the general and any other node of a given network. This answers an open question in Mazoyer (*in* "Automata Networks" (C. Choffrut, Ed.), pp. 82–93, Springer-Verlag, Berlin/New York, 1986). Our result is based on a strategy different from the one of Balzer and Waksman, which is used in almost all existing solutions of fssp and gfssp. The extension of gfssp and gfssp-nud to networks with more than one general is also considered. We show that (1) for any fixed $k \geq 2$, gfssp with at most k generals has a solution whose firing time is bounded by $O(D)$, where D is the maximum distance between any two nodes of a given network, and gfssp-nud with at most k generals has a solution whose firing time is bounded by $O((\Phi + \tau_{\max})^3)$, where Φ is the maximum transmission delay between any two nodes of a given network; (2) there are no solutions for gfssp and gfssp-nud with an arbitrary number of generals. © 1992 Academic Press, Inc.

1. INTRODUCTION

The firing squad synchronization problem (fssp) was first raised by Myhill (1957). We rephrase the statement of the problem as follows. Consider a finite (but arbitrarily long) linear array of identical (Moore type) finite automata (also referred to as nodes). The leftmost node of the array is called the *general* and the others are called *soldiers*. The nodes operate synchronously at discrete steps and communicate with each other

* Research supported in part by a grant from SERB, McMaster University, and NSERC Operating Grant OGP 0046613.

0890-5401/92 $3.00

Copyright © 1992 by Academic Press, Inc.
All rights of reproduction in any form reserved.

by sending/receiving signals to/from their neighbors. The state and output signals of each node at time t are a function of its own state and the output signals of its two neighbors at time $t-1$. The general can be *excited* (i.e., set to a special state *Start*) by some external means, such as an external stimulus. The problem is to specify the structure (i.e., states, output signals, and transition function) of the nodes such that the general, after being excited, can cause all the nodes to enter a special state, called *Fire*, exactly at the same time. The time required for the general to cause all the nodes to enter *Fire* is called the *firing time*.

A first solution of fssp was given by McCarthy and Minsky (Minsky, 1967). Solutions with minimum firing time were discovered by Goto (Moore, 1964), Balzer (1967), and Waksman (1966). These latter solutions all have firing time $2n-2$ for arrays of length n, but the solutions of Balzer and Waksman use many fewer states than the one of Goto. Balzer's and Waksman's solutions are based on the same (divide-and-conquer) strategy which is to repeatedly break a given array into two equal parts through the "collision" of signals with different speeds, until single nodes are obtained (Balzer, 1967; Waksman, 1966). This strategy was widely used in later researches on fssp.

Note that, in the above definition of fssp, we assumed that there is no transmission delay involved; i.e., it takes 0 time for an output signal of a node to go through a link and reach a neighbor of the node. Varshavsky *et al.* (1970) considered a variation of fssp in which transmission of a signal via a link requires τ time for some integer $\tau \geqslant 0$; i.e., all links have the same delay τ. A solution to such a problem should be independent of the actual value of τ. Denote this problem by fssp-ud (i.e., fssp with uniform delays). Varshavsky *et al.* (1970) presented a solution of fssp-ud whose firing time is $(\tau+1)^2+(\tau+1)(2n-2)$ for n-node arrays with delay τ on links. The basic idea is to generate "gating" signals with period $\tau+1$ in the array so that the nodes change their states synchronously once every $\tau+1$ time and then carry out the Balzer–Waksman strategy at a speed $1/(\tau+1)$. It is natural to generalize fssp-ud so that the transmission delays of links are not necessarily all the same. Denote this problem by fssp-nud (i.e., fssp with nonuniform delays). It was an open question whether there exists a solution of fssp-nud (Mazoyer, 1986). (Again, the solution should be independent of the actual delays.)

We answer the open question in the positive. In fact, we give a solution to a more general problem described below.

One can extend fssp to arbitrarily connected networks and allow the general to be anywhere in a network. This generalized fssp (denoted by gfssp) was extensively studied in (Kobayashi, 1978a, 1978b; Nishitani and Honda, 1977; Romani, 1976; Rosenstiehl, 1973). (For discussions of some less general variations of fssp, see (Culik, 1989; Grasselli, 1975; Kobayashi,

1977; Moore and Langdon, 1968; Romani, 1978; Shinahr, 1974; Szwerinski, 1982).) Rosenstiehl *et al.* (1973) first obtained a solution of gfssp whose firing time is $2n$ for networks of n nodes. The basic idea is to construct a spanning tree for a given network and form a linear array that include all nodes of the tree (thus, all nodes of the network), using the "snaking" technique. The solution of Balzer or Waksman is then simulated in the linear array. Romani (1976) used a similar idea and gave a solution whose firing time is better than $2n$ for some networks. Nishitani and Honda (1977) took a different approach and obtained a $4R$ time-bounded solution, where R is the radius of a given network, i.e., the maximum distance between the general and any other node of the network, assuming that the distance between neighbors is 1. (By a $4R$ time-bounded solution we mean a solution whose firing time is at most $4R$ for networks of radius R.) All of these solutions use Balzer–Waksman strategy at their final stages.

Let gfssp-nud denote the extension of gfssp to networks with nonuniform delays on links. Clearly, fssp-nud is a special case of gfssp-nud. We present an $O(\Delta^3 + \tau_{\max})$ time-bounded solution of gfssp-nud, where $\tau_{\max}$ is the maximum delay of any single link and Δ is the maximum delay between the general and any other node of a given network, i.e., $\Delta = \max\{\min\{j - 1 + \sum_{i=1}^{j} \tau(e_i) \mid e_1 \cdots e_j \text{ is a path from the general to } u, \tau(e_i) \text{ is the delay of link } e_i,\ 1 \leqslant i \leqslant j\} \mid u \text{ is a node of the network}\}$. The parameter Δ of a network is called the *delayradius* of the network.

It is easy to see that Balzer–Waksman strategy and its variations (e.g., the one in Varshovsky *et al.*, 1970) do not work for fssp-nud (hence, gfssp-nud). The reason is that, since the delays are nonuniform, generally a linear array cannot be broken into two equal parts. Our solution of gfssp-nud adopts a different strategy. Although our strategy is also to iteratively break a given network into single nodes, in each iteration, it breaks the network currently under consideration into pieces that have approximately the same delayradii. The details of the strategy will be given in Section 3.

We also consider gfssp and gfssp-nud with two or more generals. The generals of a network can be excited at different times. If at least one general is excited, then all nodes of the network should enter *Fire* at the same time. This corresponds to the case that in a (biological/computer) system, two or more elements may initiate the synchronization process around (not necessarily exactly) the same time. Define the *diameter* D (*delaydiameter* Φ) of a network to be the maximum distance (delay, respectively) between any two nodes of the network. We show that (1) for any fixed $k \geqslant 2$, gfssp with at most k generals has an $O(D)$ time-bounded solution and gfssp-nud with at most k generals has an $O((\Phi + \tau_{\max})^3)$ time-bounded solution; (2) there are no solutions for gfssp or gfssp-nud with arbitrary number of generals.

The paper is organized as follows. Section 2 gives the formal definition of gfssp-nud and some basic terms and notation. In Section 3, we first informally explain how a node can store (unbounded) numbers by circulating signals along a circuit. Then we give the basic idea behind our solution of gfssp-nud and a brief discussion of its implementation. In Section 4, we describe an equivalent formulation of circuits. Using this formulation, we show that some interesting arithmetic and conversion operations can be done on the numbers stored on a circuit. These results enable us to create counters in a network. In Section 5, we construct the solution of gfssp-nud. Since we are interested only in the existence of the solution and the order of its time complexity, only a high level description of the solution will be given and we will be very generous with the constants. Section 6 is concerned with the synchronization of networks having two or more generals. A brief discussion on further improvement of the results is given in Section 7.

2. Preliminaries

Throughout the paper, let $d \geqslant 2$ be a fixed integer. A *network of finite automata*, or simply a *network*, is a connected weighted (undirected) graph with degree at most d whose nodes represent copies of a finite automaton (fa) A and whose links (i.e., edges) represent communication lines between the fa's. (The degree of a graph is the maximum degree of its nodes.) For simplicity, we will not distinguish between a node and the fa represented by the node. One node is distinguished from the others and is called the *general*. The weight of a link e, denoted by $\tau(e)$, is any nonnegative integer and is referred to as the *delay* of e.

The fa A consists of a finite control and d input/output terminals. The terminals of A are numbered from 1 to d. A receives/sends signals through these terminals. The signal that A receives (sends) throgh its ith terminal at time t is called the ith input (output) signal of A at time t. If the state and the input signals of A at time t are $q, i_1, ..., i_d$, then the state and output signals $p, o_1, ..., o_d$ of A at time $t+1$ are given by the following formula: $(p, o_1, ..., o_d) = f_A(q, i_1, ..., i_d)$. The function f_A is called the *transition function* of A. Note that f_A involves a delay of one time step. A state *Quiet* of A is designated as the *quiescent state* and a signal \$ of A is designated as the *quiescent signal*. For convenience, we also assume that A has a dummy signal λ whose purpose will be given below. If the current state of A is *Quiet* and the current input signals are \$'s or λ's, then A remains in state *Quiet* and outputs \$'s, i.e., $f_A(Quiet, i_1, ..., i_d) = (Quiet, \$, ..., \$)$ if each i_j is either \$ or λ. A is said to be quiescent if its state is *Quiet* and its input signals are \$'s or λ's.

For each node u (recall that u is a copy of A), the links incident on u and the terminals of u are connected as follows: each link is connected to a terminal and every terminal is connected to at most one link. We say that a terminal is *occupied* if it is connected to some link. The terminals not connected to any links are said to be *open*. If the ith terminal of u is connected to a link (u, v), then v is called the ith neighbor (or neighbor i) of u.

During a computation, the nodes of the network operate synchronously at discrete time steps. They communicate with each other by sending/receiving signals through the terminals and links. The input signals of a node u are determined as follows. If the ith terminal of u is open, then the ith input signal of u is λ (i.e., the dummy signal). Otherwise if the ith terminal of u is connected to the jth terminal of another node v through the link (u, v), then the ith input signal of v at time t is the jth output signal of u at time $t-\tau((u, v))$.

Figure 1 illustrates an example of networks. In the figure, the integer above the middle of a link e is the delay of e (i.e., $\tau(e)$) and the integers around a node u indicates how the links incident on u are connected to the terminals of u (the integers are the terminal indices). Note that, although it is not explicitly shown in the figure, each node of the network involves a one-step delay.

The generalized firing squad synchronization problem with nonuniform delays (gfssp-nud) is to construct an fa A with the following property. Let N be any network whose nodes are copies of A. Initially, all the nodes of N are quiescent. Suppose that, at some time t_0, the general of N is set (by some external means) to a special state *Start*; i.e., the general is excited at time t_0. Then there exists a moment t_1 such that all the nodes of N enter a special state *Fire* exactly at time t_1. (We assume that the external

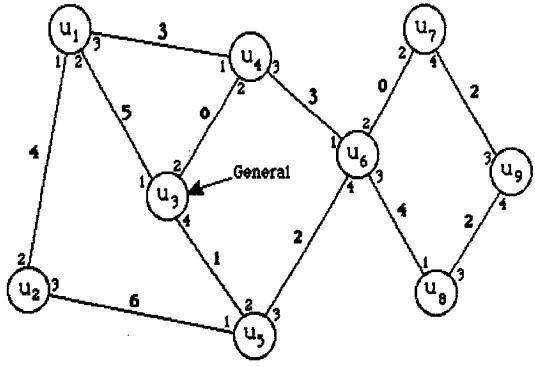

FIG. 1. A network (assuming $d=4$).

excitation may happen only when the general is quiescent. Therefore, the network N is left alone by the external world after the general is excited.)

Such an A is called a *solution* of gfssp-nud and the time $t_1 - t_0$ is called the *firing time* of solution A on network N. The generalized firing squad synchronization problem (gfssp) is a special case of gfssp-nud in which we assume that the delays of links are 0. The restriction of gfssp-nud to tree-structured networks (or trees for short) is denoted by tfssp-nud. Similarly, the restriction of gfssp to trees is denoted by tfssp.

Let N be a network and G be the general of N. The following terms and notation are needed.

We say that N *fires at time* t if all nodes of N enter *Fire* exactly at time t. When G makes N fire, we also say that G *synchronizes* N.

A *path* is an *ordered* sequence $e_1 \cdots e_m$ of links such that $e_1 = (u_0, u_1), ..., e_m = (u_{m-1}, u_m)$ for some nodes $u_0, u_1, ..., u_{m-1}, u_m$. A path $(u_0, u_1) \cdots (u_{m-1}, u_m)$ is also denoted by $u_0 u_1 \cdots u_{m-1} u_m$. The reverse of a path p is denoted by p^R and the concatenation of two paths p_1 and p_2 is denoted by $p_1 p_2$. A path p is said to be *simple* if it passes through each node at most once. A *circuit* is a path that begins and ends at the same node. In this paper, we are only interested in circuits which can be expressed as pp^R for some simple path p. Note that we distinguish circuits having different beginning nodes. For any path $p = e_1 \cdots e_m$ of N, the *length* of p, denoted by $|p|$, is m, and the *delay* of p, denoted by $\tau(p)$, is $m - 1 + \sum_{i=1}^{m} \tau(e_i)$. Clearly, $\tau(p)$ is the time needed for a signal to go through p. For any two nodes u_1 and u_2, the *distance between* u_1 and u_2 is $\min\{|p| \mid p \text{ is a path from } u_1 \text{ to } u_2\}$ and the *delay between* u_1 and u_2 is $\min\{\tau(p) \mid p \text{ is a path from } u_1 \text{ to } u_2\}$. The *radius* R of N is $\max\{$the distance between G and $u \mid u$ is a node of $N\}$. The *diameter* D of N is $\max\{$the distance between u_1 and $u_2 \mid u_1$ and u_2 are nodes of $N\}$. The *delayradius* Δ of N is $\max\{$the delay between G and $u \mid u$ is a node of $N\}$. The *delaydiameter* Φ of N is $\max\{$the delay between u_1 and $u_2 \mid u_1$ and u_2 are nodes of $N\}$. Clearly, $D \leqslant 2R$ and $\Phi \leqslant 2\Delta + 1$. A *long path* of N is any simple path that begins at the general and has delay Δ. The *maximum link delay* of N, denoted by $\tau_{\max}$, is $\max\{\tau(e) \mid e \text{ is a link of } N\}$.

A *subnetwork* of N (e.g., a spanning tree) is a network which can be obtained by removing some nodes and links from N. Note that a subnetwork has its own designated general. A *partition* of N is any set of disjoint subnetworks of N that include all nodes of N. A *minimum-delay spanning tree* Π of N is a spanning tree of N such that (1) G is the general of Π and (2) for any node u, the delay between G and u in Π is the same as the delay between G and u in N. Clearly, the delayradius of a minimum-delay spanning tree of N is Δ.

During a computation, we say that a subnetwork N' is *marked* in N (or simply *marked*) if for each node u of N, if u belongs to N', then u knows

which of its neighbors in N are also its neighbors in N'. Thus, if a subnetwork N' is marked (in N), then the computation of N' can be simulated by the nodes of N. By to *construct* a subnetwork we mean to have the subnetwork marked. By to *break* N into a set of subnetworks we mean to have each of the subnetworks marked.

3. Basic Idea

First, we describe how a node can store (unbounded) numbers by utilizing signals. This technique is one of the keys to our solution of gfssp-nud.

Let $p = u_0 u_1 \cdots u_m$ be a simple path of network N. Suppose that the path p is marked (during a computation). Then node u_0 can send signals to itself via circuit pp^R by asking nodes $u_1, ..., u_m$ to "cooperate," i.e., each u_i should pass the signal received from u_{i-1} on to u_{i+1} and the signal received from u_{i+1} on to u_{i-1}, $1 \leqslant i < m$, and u_m should send the signal received from u_{m-1} back to u_{m-1}. u_0 can ask for this "cooperation" by propagating a request (signal) along path p. We assume that such a request is always "approved" by $u_1, ..., u_m$. When u_0 propagates such a request along p, we say that u_0 *forms* a circuit pp^R and, in the subsequent steps, pp^R is called a circuit *formed* by u_0. When u_0 propagates a signal along p to "cancel" the request, we say that u_0 *releases* circuit pp^R. After pp^R is released, it is no longer referred to as a formed circuit. Note that, the above definition (and assumption) imply that a link cannot be shared by two different formed circuits at a same time. But since a link can be divided into a constant number of "conceptual links" (by using composite signals), we can generalize the definition so that a link can be shared by a constant number of formed circuits at a same time.

Suppose that the node u_0 forms the circuit pp^R at time t_0 and, starting from t_0, u_0 sends signals to itself via pp^R. When u_0 receives/sends the signal s from/to u_1, we also say that u_0 receives/sends s from/onto circit pp^R. Define $\theta(p) = \tau(pp^R) + 1 = 2\tau(p) + 2$. Consider the signals sent by u_0 to itself via pp^R. As these signals travel through pp^R, they form a *signal queue* Q. When u_0 sends a signal into pp^R, the signal is added to Q from the rear. When u_0 receives a signal from pp^R (after time $t_0 + \theta(p) - 1$), the signal is removed from the front of Q. The size of Q increases at unit speed (i.e., one per step) from time t_0 to time $t_0 + \theta(p) - 1$. After $t_0 + \theta(p) - 1$, Q is of constant size $\theta(p)$. Clearly, this signal queue Q can be used as a storage and node u_0 can store information such as numbers (represented as sequences of digits) in it. For convenience, we refer to the numbers stored in Q as numbers *stored on circuit* pp^R. We defer more discussion about circuits (as storages) to the next section. In the next section, we show that it is possible to perform arithmetic and conversion operations and count using circuits.

Now we can give the basic idea behind our solution of gfssp-nud. Denote our intended solution of gfssp-nud by A_{gnud}. Similar to the existing solutions of gfssp, A_{gnud} also works in two stages for a given network: (1) a minimum-delay spanning tree of the network is constructed and (2) the tree is made to fire. It is not hard to show that for any network N with delayradius Δ and maximum link delay $\tau_{\max}$, a minimum-delay spanning tree of N can be constructed in $2\Delta + 2\tau_{\max} + 4$ steps. The principle of the construction is very similar to the one of finding rooted trees of minimal paths given in (Moore, 1959; Rosenstiehl *et al.*, 1973). In order not to introduce too many details in this section, we defer the details of the construction to Section 5. For now, we just assume that such a construction exists.

Since the first stage of A_{gnud} can be done in $2\Delta + 2\tau_{\max} + 4$ time, it suffices to show that the second stage can be done in $O(\Delta^3)$ time. That is, we only need to give an $O(\Delta^3)$ time-bounded solution of tfssp-nud. In fact, we can construct a solution of tfssp-nud whose firing time is *exactly* $f(\Delta)$ for trees with delayradius Δ, where $f(\Delta) = O(\Delta^3)$ is some increasing function. Let A_{tnud} denote our intended solution of tfssp-nud. In the rest of this section, we describe the basic idea behind the construction of A_{tnud} and discuss briefly how it can be implemented.

For convenience, we think of the trees as rooted trees. The root of a tree is its general. From now on, it should be understood that the nodes of all considered trees are copies of A_{tnud}, unless otherwise stated. Thus, by saying that a node u (of some tree) can do job X, we actually mean that A_{tnud} is so constructed that u can do job X.

The basic idea is as follows. Let $f(\Delta)$ be the function mentioned above; i.e., we want $f(\Delta)$ to be the firing time of A_{tnud} for trees of delayradius Δ. Clearly, $f(\Delta) = \Omega(\Delta)$. For any $\Delta_1 \geqslant \Delta_2$, define $g(\Delta_1, \Delta_2) = f(\Delta_1) - f(\Delta_2)$. (Note that $f(\Delta)$ is increasing in Δ.) Let Π be the tree under consideration, G be its root (i.e., the general), and Δ be its delayradius. Without loss of generality, assume that G is excited at time 0. If $\Delta = 0$ (i.e., G is the only node of Π), then G enters *Fire* in $f(0)$ steps. Otherwise, suppose that $\Delta > 0$. Let $\{\Pi_1, ..., \Pi_k\}$ be a partition of Π such that the delayradius of each Π_i is less than Δ. Let G_i and Δ_i be the root and delayradius of Π_i, respectively, $1 \leqslant i \leqslant k$. Suppose that G has a way to break Π into the subtrees $\Pi_1, ..., \Pi_k$ and locate the nodes $G_1, ..., G_k$. Then, to make Π fire at time $f(\Delta)$, G only needs to break Π into $\Pi_1, ..., \Pi_k$ and make sure that G_i begins to repeat the process just described in Π_i exactly at time $g(\Delta, \Delta_i)$, for each $i = 1, 2, ..., k$.

It is easy to see that, if the above idea can be successfully applied to Π and all its subtrees, then Π fires at time $f(\Delta)$.

The following are some thoughts on the implementation of this idea. The construction for the case $\Delta = 0$ is trivial. So we assume $\Delta > 0$. We have to

decide how to choose the partition $\{\Pi_1, ..., \Pi_k\}$. Since the nodes are only fa's, the subtrees $\Pi_1, ..., \Pi_k$ should be chosen such that it is easy for G to break Π into these subtrees. Also, it should be easy for G to locate the nodes $G_1, ..., G_k$. Another key to the success of the above idea is that each G_i should maintain a counter C_i so it knows to begin its own synchronization process exactly at time $g(\Delta, \Delta_i)$. The counter C_i $(1 \leqslant i \leqslant k)$ can be realized by constructing a simple path p_i beginning at G_i, forming the circuit $p_i p_i^R$, storing the number $g(\Delta, \Delta_i)$ on pp^R, and then counting from $g(\Delta, \Delta_i)$ down to 0. (This is only a rough idea. The actual construction given in Section 5 is a bit different from this scenario. The details concerning how to count using the numbers on a circuit will be given in next section.) The paths $p_1, ..., p_k$ will be referred to as *counting paths.* Since the circuit $p_i p_i^R$ has to hold the number $g(\Delta, \Delta_i)$, $\theta(p_i) = \Omega(\log g(\Delta, \Delta_i))$, $1 \leqslant i \leqslant k$. After the synchronization process in a subtree Π_i is started, the counter C_i becomes useless and thus the circuit $p_i p_i^R$ can be released. Note that circuits $p_1 p_1^R, ..., p_k p_k^R$ may be released at different times. In order to make the synchronizations of these subtrees independent of each other, the path p_i should not contain any links of Π_j, for any $j \neq i$. (Otherwise, there could be a chance that as Π is broken into more and more pieces, an unbounded number of formed circuits need to go through a same link at a same time. This is impossible to implement because a link can only be shared by a constant number of formed circuits simultaneously.)

With these requirements in mind, we propose the following so-called delayradius-reduction scheme (DRRS) for choosing the partition. It will be shown that if DRRS is followed when choosing the partition, then for each member Π_i of the partition, we can find a counting path p_i for Π_i such that (1) $\theta(p_i) \geqslant \Delta + 1$ and (2) p_i does not overlap with any other members of the partition. Since $f(\Delta) = O(\Delta^3)$, clearly $\theta(p_i) = \Omega(\log f(\Delta)) = \Omega(\log g(\Delta, \Delta_i))$.

Let Π be a tree with root G and delayradius $\Delta > 0$. We want to choose a partition of Π whose members all have delayradii less than Δ and have desirable counting paths. First, we describe the intuition behind DRRS. Let $G_1, ..., G_k$ be the children of G and $\Pi_1, ..., \Pi_k$ be the corresponding subtrees of G. (Clearly, $k \leqslant d$.) Let Δ_i be the delayradius of Π_i and $\tau_i = \tau((G, G_i))$, $1 \leqslant i \leqslant k$. Without loss of generality, assume that $\Delta_i + \tau_i \geqslant \Delta_{i+1} + \tau_{i+1}$, $1 \leqslant i < k$. Let j be the index such that $\Delta_i + \tau_i + 1 = \Delta$, $1 \leqslant i \leqslant j$, and $\Delta_i + \tau_i + 1 < \Delta$, $j < i \leqslant k$. Finally, let Π' be the subtree obtained from Π by pruning $\Pi_1, ..., \Pi_j$ and Δ' be the delayradius of Π'. (See Fig. 2(a).)

Let us examine the subtrees $\Pi_1, ..., \Pi_j$, and Π' and see which one can be a member of the partition. Clearly, $\Delta_i = \Delta - \tau_i - 1 < \Delta$, $1 \leqslant i \leqslant j$, and $\Delta' = \max\{0\} \cup \{\Delta_i + \tau_i + 1 \mid j < i \leqslant k\} < \Delta$. Let i be any index between 1 and j. Since $\Delta_i + \tau_i + 1 = \Delta$, it is easy to see that either a long path of Π_i or $G_i G$ is a desirable counting path for Π_i, depending on whether $\Delta_i \geqslant \tau_i$.

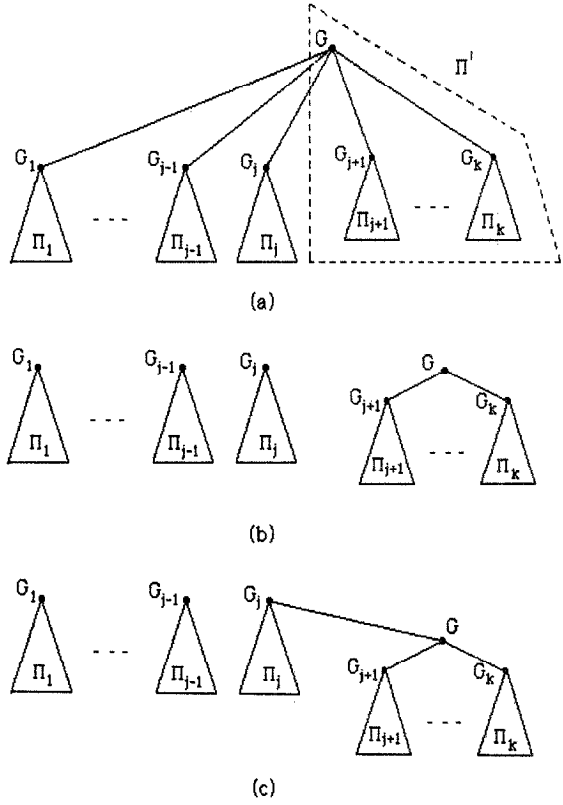

FIG. 2. Scheme DRRS.

Thus, Π_i can be a member of the partition. Similarly, if $\Delta' + \tau_j + 1 \geqslant \Delta$, then either a long path of Π' or GG_j is a desirable counting path for Π' and thus Π' can be a member of the partition. However, if $\Delta' + \tau_j + 1 < \Delta$, then a desirable counting path for Π' may not exist. In this case, let Π_j' denote the tree obtained by joining Π' and Π_j with link (G, G_j) and making G_j the new root. Let Δ'_j be the delayradius of Π_j'. Then, $\Delta_j' = \max\{\Delta' + \tau_j + 1, \Delta_j\} < \Delta$. Also, it is easy to see that a long path of Π_j' is a desirable counting path for Π_j'. Hence, we can make Π_j' a member of the partition.

Now we can formally present DRRS. The partition is chosen according to the following two cases.

DDRS. *Case* 1. $\Delta' + \tau_j + 1 \geqslant \Delta$. Choose $\{\Pi_1, ..., \Pi_j, \Pi'\}$ as the partition. (See Fig. 2(b).)

Case 2, $\Delta' + \tau_j + 1 < \Delta$. Let Π_j' be the tree defined above. Choose

$\{\Pi_1, ..., \Pi_{j-1}, \Pi_j'\}$ as the partition. See Fig. 2(c).) Note that, when $j=1$, the partition only contains Π_j'' which is rerooted Π.

Let P denote the partition chosen by DRRS for the tree Π. Then the counting paths for the members of P can be found as mentioned above. There are two cases.

Case 1. $P=\{\Pi_1, ..., \Pi_j, \Pi'\}$. Let p_i if $\Delta_i \geqslant \tau_i$, or G_iG otherwise, $1 \leqslant i \leqslant j$, and let p' be a long path of Π' if $\Delta' \geqslant \tau_j$, or GG_j otherwise. Choose $p_1, ..., p_j, p'$ as the counting paths for $\Pi_1, ..., \Pi_j, \Pi'$, respectively. Note that, $\theta(p_i)=2\tau(p_i)+2 \geqslant \Delta_i+\tau_i+2=\Delta+1$, $1 \leqslant i \leqslant j$. Also, since $\Delta'+\tau_j+1 \geqslant \Delta$, $\theta(p')=2\tau(p')+2 \geqslant \Delta'+\tau_j+2 \geqslant \Delta+1$.

Case 2. $P=\{\Pi_1, ..., \Pi_{j-1}, \Pi_j'\}$. The counting paths $p_1, ..., p_{j-1}$ are the same as in Case 1. Let p_j' be a long path of Π_j'. Choose p_j' as the counting path for Π_j'. Clearly, $\theta(p_j')=2\tau(p_j')+2=2\Delta_j'+2 \geqslant \Delta_j+\Delta'+\tau_j+3 \geqslant \Delta+2$.

It is easy to see that, if the counting paths are always chosen as described above, then during the synchronization of a tree, each link of the tree is shared by at most two counting paths (hence, by at most two formed circuits) at any time.

Now we consider how to implement DRRS and construct the counting paths as described above in a tree. Again, let us consider the tree Π. In order to break Π into subtrees according to DRRS and construct the corresponding counting paths, G needs to test conditions such as $\Delta_i+\tau_i+1 \geqslant \Delta_{i+1}+\tau_{i+1}+1$, $\Delta_i+\tau_i+1 > \Delta_{i+1}+\tau_{i+1}+1$, $\Delta'+t_j+1 \geqslant \Delta$, $\Delta_i \geqslant \tau_i$, $\Delta' \geqslant \tau_j$, etc., and construct a long path in each of $\Pi_1, ..., \Pi_j, \Pi'$, and Π_j'. These can be done by using a simple technique. We illustrate the technique by proving a general lemma.

LEMMA 1. *Let π_1 and π_1 be two trees rooted at a same node H and δ_1 and δ_2 be their delayradii, respectively. Suppose that, besides H, π_1 and π_2 have no other common nodes. Then* (1) *H can construct a long path of π_1 in $2\delta_1+2$ steps*; (2) *H can check if $\delta_1 \geqslant \delta_2$ in $2\min\{\delta_1, \delta_2\}+2$ steps.*

Proof. (1) H constructs a long path of π_1 as follows. H sends a signal # to all of its children in π_1 simultaneously. The signal # is propagated downward (along all paths) until it reaches the leaves of π_1. When a leaf receives a # from its parent, it sends a # back. The # is then propagated upward. While propagating #'s upward, each internal node observes from which child it receives the last # and remembers the child (if there is more than one such child, it arbitrarily chooses one). When H receives the last # from its children in π_1, a long path of π_1 is naturally constructed. (2) The condition $\delta_1 \geqslant \delta_2$ can be tested using the same technique. I.e., H propagates a signal down and up simultaneously (and independently) in both π_1 and π_2, and observes in which of π_1 and π_2 the propagation is

completed first. $\delta_1 \geqslant \delta_2$ if and only if the propagation is completed first in π_2 (including the case when both trees finishes the propagation at the same time). ▮

Thus, G can test the conditions $\Delta_i + t_i + 1 \geqslant \Delta_{i+1} + t_{i+1} + 1$ and $\Delta_i + t_i + 1 > \Delta_{i+1} + t_{i+1} + 1$, $1 \leqslant i < k$, in $2\Delta + 2$ steps. To test the condition $\Delta_i \geqslant \tau_i$, $1 \leqslant i \leqslant k$, G sends G_i a signal asking it to test the condition and report the result back. This takes $2\tau_i + 2 + 2\min\{\tau_i, \Delta_i\} + 2 \leqslant 2\Delta + 2$ steps. Clearly, after j is found, the condition $\Delta' \geqslant \tau_j$ can be tested in $2\Delta + 2$ steps. To test the condition $\Delta' + \tau_j + 1 \geqslant \Delta$, G propagates a signal down and up in Π and, at he same time, propagates the same signal down and up in Π' and GG_j, sequentially (i.e., when the propagation in Π' is completed, it starts the propagation in GG_j). Then it checks whether the propagation is completed first in Π or in GG_j. This takes at most $2\Delta + 2$ steps. Hence, G can test all hese conditions in a total of $4\Delta + 4$ steps.

Once G has tested all the conditions, it can construct the required long paths. The long path p' can be constructed directly in $2\Delta + 2$ steps. To construct p_i, $1 \leqslant i \leqslant j$, G sends G_i a signal asking it to construct p_i and report the completion of the construction back. This takes $2\tau_i + 2 + 2\max\{\tau_i, \Delta_i\} + 2 \leqslant 4\Delta + 4$ steps. Similarly, p'_j can be constructed in $2\tau_j + 2 + 2\Delta'_j + 2 \leqslant 4\Delta + 4$ steps. Also, since G knows what subtrees should be chosen according to DRRS, the actual breaking of Π into those subtrees can be done in $2\Delta + 2$ steps. Thus, G can break tree Π according to DRRS and construct the corresponding counting paths in a total of $8\Delta + 8$ steps.

4. Computing and Counting on Circuits

In this section, we show how to compute and count using circuits. Here, we consider only natural numbers (i.e., nonnegative integers). We assume that the numbers are in radix b, where $b > 1$ is some constant whose exact value will be determined in Section 5. The unary representation of a number x is 1^x. Note that the unary representation of 0 is the null string. A string 1^x $(x \geqslant 0)$ is also called a unary number. The length of a number x, denoted by $|x|$, is the number of digits that x has, i.e., $|x| = \lfloor \log_b x \rfloor + 1$. The length of a unary number 1^x, denoted by $|1^x|$, is x.

To facilitate the discussion, we give an equivalent formulation of circuits. Let u be a node (of some tree) and p be a simple path beginning at u. Suppose that u forms the circuit pp^R at time t_0. We imagine that u owns a circular tape T consisting of $\theta(p)$ cells. Tape T has a read/write head H that shifts along the tape clockwise at unit speed (i.e., one cell per step). When head H is at a cell i, u can read and overwrite the contents of cell i. The cells are indexed from 0 to $\theta(p) - 1$ clockwise. At time t_0, H is at cell

0 and all cells contain blanks. For simplicity, we let cell 0 be marked by some special symbol so u always knows if H is scanning cell 0. As far as u is concerned, the circuit pp^R can be replaced by the tape T as follows: instead of sending/receiving signals onto/from pp^R, u writes/reads symbols on/from tape T. It is easy to see that, for node u, having tape T is equivalent to forming circuit pp^R. Thus, we can assume that, whenever u forms the circuit pp^R, it automatically obtains the tape T. Since circular tapes are very similar to conventional storages (e.g., the worktapes of Turing machines), we will present the results in terms of circular tapes instead of circuits. But the reader should always be aware that the numbers are actually stored on circuits and all the operations are actually performed using circuits.

Let u be a node and p be a simple path beginning at u. Suppose that u has formed circuit pp^R. Let T be the corresponding imaginary tape of u and H be the read/write head. Suppose that T can (conceptually) be divided into sufficiently many tracks (by using composite symbols). The (unary) numbers are stored on T as follows. Each (unary) number is stored in a different track of T. The digits of a number (or the bits of a unary number) are positioned counterclockwise with the least significant digit (or the last bit) at cell 0. Thus, if $a_m a_{m-1} \cdots a_1$ is an m-digit number (or an m-bit unary number) stored on T, where a_m and a_1 are the most and least significant digits (or the first and last bits) respectively, then a_m is contained in cell $m-1$, a_{m-1} is contained in cell $m-2, \ldots, a_1$ is contained in cell 0. Clearly, only (unary) numbers of length at most $\theta(p)$ can be stored on T.

In the following lemmas and corollary, we assume that when a computing or counting process is started, the head H is canning cell 0 and the given (unary) numbers (if there is any) are all stored on T and when the process is finished, the given (unary) numbers are intact and the result (if there is one) isstored on T. Thus, by "compute" we actually mean "compute and store." We also assume that the arithmetic and conversion operations do not result in negative numbers or (unary) numbers with lengths $>\theta(p)$.

Lemma 2 shows that arithmetic operations such as addition, subtraction, multiplication, integer division, and mod can be done using tape T.

LEMMA 2. *Given numbers x_1 and x_2, u can compute*

(1) $c_1x_1 + c_2x_2 + c_3$ *in* $\theta(p)$ *steps, for any constant* c_1, c_2, *and* c_3,

(2) x_1x_2 *in* $\theta(p)^2$ *steps,*

(3) $\lfloor x_1/x_2 \rfloor$ *in* $3\theta(p)^2$ *steps,*

(4) $x_1 \bmod x_2$ *in* $3\theta(p)^2$ *steps.*

Proof. In the following, let $r = |x_1|$ and $s = |x_2|$. It is convenient to divide a computation of u into *cycles*. A cycle consists of $\theta(p)$ steps.

(1) Let $y = c_1x_1 + c_2x_2 + c_3$. u can compute y in one cycle. In the cycle, while head H scans the cells 0, 1, ..., $\theta(p)-1$, u reads x_1 and x_2, digit by digit with the least significant digit first, and determines y, also digit by digit with the least significant digit first, by performing the specified addition.

(2) Let $x_2 = a_s \cdots a_1$. The computation has s cycles. In the first cycle, u computes $y_1 = a_1x_1$ and $y_2 = bx_1$, writes a marker $\downarrow$ into cell 1, and remembers a_2 if it exists (i.e., if $s > 1$). If $s = 1$, u ends the computation. Obviously, $y_1 = x_1x_2$. Now suppose $s > 1$. In the ith cycle, $2 \leqslant i \leqslant s-1$, u increments y_1 by a_iy_2, multiplies y_2 by b, shifts the marker $\downarrow$ one cell clockwise, and remembers the digit a_{i+1}. In the sth cycle, u increments y_1 by a_sy_2. Note that, since the marker $\downarrow$ is at cell i in the ith cycle, u can recognize the digit a_{i+1}, $1 \leqslant i \leqslant s-1$. Clearly, at the end of the sth cycle, $y_1 = x_1x_2$.

(3) and (4) In the first cycle, u checks whether $s < r$ and sets $y_1 = x_1$, $y_2 = x_2$, and $y_3 = 1$. If $s \geqslant r$, u uses the 2nd cycle to determine the largest digit c_1 such that $c_1x_2 \leqslant x_1$. In the 3rd cycle, u sets $y_4 = c_1$ and $y_5 = x_1 - c_1x_2$ and ends the computation. Obviously, $y_4 = \lfloor x_1/x_2 \rfloor$ and $y_5 = x_1 \bmod x_2$. Otherwise, suppose $s < r$. u multiplies y_2 by b and increments y_3 by 1 every cycle until the most significant digit of y_2 is aligned with the 2nd most significant digit of y_1. This takes $r-s-1$ cycles. In the $(r-s+1)$st cycle, u determines the largest number z such that $zy_2 \leqslant y_1$. Clearly, $z < b^2$. Starting fro the $(r-s+2)$nd cycle, u decrements y_3 by 1 every cycle until it equals 0. Clearly, y_3 is decremented to 0 in the $(2r-2s+1)$st cycle. Meanwhile, u does the following. In the $(r-s+2)$nd cycle, u computes $y_1 = b(y_1 - zy_2)$, $y_4 = z$, and $y_5 = b$, and determines the largest digit c_{r-s-1} such that $c_{r-s-1}y_2 \leqslant y_1$. In the ith cycle, $r-s+3 \leqslant i \leqslant 2r-2s+1$, u computes $y_1 = b(y_1 - c_{2r-2s-i+2}y_2)$, $y_4 = by_4 + c_{2r-2s-i+2}$, and $y_5 = by_5$, and determines the largest digit $c_{2r-2s-i+1}$ such that $c_{2r-2s-i+1}y_2 \leqslant y_1$. At the end of the $(2r-2s+1)$st cycle, $y_4 = \lfloor x_1/x_2 \rfloor$, $y_5 = b^{r-s}$, and $y_1 = (x_1 \bmod x_2)b^{r-s}$. Note that, in each of the above cycles, $|y_1| \leqslant |by_2| = s+1 \leqslant r \leqslant \theta(p)$, $|y_4| \leqslant |\lfloor x_1/x_2 \rfloor| \leqslant r \leqslant \theta(p)$, and $|y_5| \leqslant r-s+1 \leqslant r \leqslant \theta(p)$. Then u computes $y_6 = \lfloor y_1/y_5 \rfloor = x_1 \bmod x_2$, using the technique described above. This takes at most another $\max\{2r-2(r-s+1)+1, 3\} = \max\{2s-1, 3\} \leqslant 2s+1$ cycles.

Thus, u can obtain $y_4 = \lfloor x_1/x_2 \rfloor$ and $y_6 = x_1 \bmod x_2$ in a total of $2r-2s+1+2s+1 = 2r+2 \leqslant 3\theta(p)$ cycles $= 3\theta(p)^2$ steps. (Note that $\theta(p) \geqslant 2$.) ∎

It is easy to generalize (1) of Lemma 2 so that the linear combination of several (maybe more than two) numbers can be computed in $\theta(p)$ steps.

I.e., given numbers $x_1, ..., x_k$, u can compute $c_1x_1 + \cdots + c_kx_k + c_{k+1}$ in $\theta(p)$ steps for any constants k, $c_1, ..., c_k$, c_{k+1}.

The next lemma shows that u can convert a number to its corresponding unary representation or vice versa.

LEMMA 3. (1) *Given a number x, u can compute the corresponding unary number 1^x in $\theta(p)^2$ steps.*

(2) *Given a unary number x, u can compute the corresponding number $|x|$ in $\theta(p)^2$ steps.*

Proof. We prove only (1). The proof of (2) is just the "reverse" of the proof of (1). Again, we divide the computation of u into cycles.

In the first cycle, u decides if $x = 0$ and, at the same time, sets the number $x_1 = x$ and the unary numbers $y_1 = 1^0$ (i.e., the null string) and $y_2 = 1^1$. If $x = 0$, then u ends the computation and $y_1 = 1^x$. Suppose that $x > 0$. Then, u subtracts one from x_1 every cycle until it equals 1. Meanwhile, for each one subtracted from x_1, u appends a bit 1 to y_2. Clearly, at the end of the xth cycle, $x_1 = 1$ and $y_2 = 1^x$. Since we assume that $x \leqslant \theta(p)$, this process takes at most $\theta(p)$ cycles $= \theta(p)^2$ steps. ∎

Since the unary representation of number $\theta(p)$ can easily be obtained, the next corollary directly follows from Lemma 3.

COROLLARY 4. *u can compute $\theta(p)$ in $\theta(p)^2$ steps.*

Now we describe how to count using tape T. Suppose that u initiates a counting process at some time t_1. (Note that we assume H is scanning cell 0 at time t_1.) We say that u can *count* t if, in the steps subsequent to t_1, u can decide whether the present time is $t_1 + t$.

LEMMA 5. *For any constant c, u can count $c\theta(p)$ and $c\theta(p)^2$.*

Proof. Suppose that u begins to count $c\theta(p)$ or $c\theta(p)^2$ at time t_1. To count $c\theta(p)$, u simply counts how many times H has passed cell 0. (Note that cell 0 is marked.) When H arrives at cell 0 for the cth time, the time is $t_1 + c\theta(p)$. To count $c\theta(p)^2$, u shifts a marker along T at a speed of $1/c\theta(p)$ cells per step (i.e., the marker is shifted one cell clockwise every $c\theta(p)$ steps). This is possible because u can count $c\theta(p)$. When the marker arrives at cell 0, the time is $t_1 + c\theta(p)^2$. ∎

LEMMA 6. *Let t be an integer such that $t \geqslant 4\theta(p)^2$. Suppose that numbers t and $\theta(p)$ are given at time t_1. Then u can count t.*

Proof. Suppose that u begins to count t at time t_1. The counting process is divided into four phases. Phase 1 consists of the first $3\theta(p)^2$

steps. In this phase, u computes $x_1 = \lfloor t/\theta(p) \rfloor$, $x_2 = t \bmod \theta(p)$, and $x_3 = 4\theta(p)$. Phase 2 consists of the next $\theta(p)^2$ steps. In this phase, u computes $x_4 = x_1 - x_3$ and $x_5 = 1^{x_2}$. Meanwhile, it observes if x_4 is 0. Since u can count $3\theta(p)^2$ and $\theta(p)^2$, u knows when Phases 1 and 2 begin or end. Then u decides whether it should skip Phase 3 and enter Phase 4 directly depending on x_4. If $x_4 = 0$, u enters Phase 4 directly. Otherwise, it first enters Phase 3. In Phase 3, u subtracts 1 from x_4 every $\theta(p)$ steps until it becomes 0. Thus Phase 3 has $t - 4\theta(p)^2 - (t \bmod \theta(p))$ steps. In Phase 4, u observes if head H is scanning cell x_2. Since the last bit of x_5 is contained in cell $x_2 - 1$, u can recognize cell x_2. When H is scanning cell x_2, the time is $t_1 + t$. ∎

Lemmas 5 and 6 provide a means of implementing counters on a circular tape. These results as well as Lemmas 2 and 3 and Corollary 4 will be used in our final construction of A_{tnud}. Note that the above lemmas and corollary can be applied only when head H is at cell 0.

5. Solution of gfssp-nud

First, we describe the solution A_{tnud} of tfssp-nud. Let the functions f and g be as follows: $f(\Delta) = a\Delta^3 + 1$ and $g(\Delta_1, \Delta_2) = f(\Delta_1) - f(\Delta_2) = a\Delta_1^3 - a\Delta_2^3$, where a is a constant yet to be decided. Let b be the constant in Section 4. We will decide the value of a and b after describing how A_{tnud} works.

Let Π be a tree with root G and delayradius Δ. Again, suppose that G is excited at time 0. A_{tnud} is constructed as follows. For convenience, we assume that each node u of Π has a conceptual self-loop link (u, u) with delay $\tau((u, u)) = 0$.

If Δ is 0, then G enters *Fire* immediately. Otherwise, suppose $\Delta > 0$. First, G constructs a long path p of Π using exactly $2\Delta + 2$ steps and forms the circuit pp^{R}. Let T be the imaginary tape corresponding to circuit pp^{R}. Then G counts $4\theta(p)^2 = (4\Delta + 4)^2$ and, at the same time, computes the number $x = \Delta = (\theta(p) - 2)/2$ (and stores x on T), breaks Π into a set of subtrees according to DRRS, and constructs a counting path for each of those subtrees as described in Section 3. Note that, since $(4\Delta + 4)^2 > 8\Delta + 8$, u has enough time to break Π into the subtrees and construct the corresponding counting paths. Let $\Pi_1, ..., \Pi_k$ be the resulting subtrees and G_i and Δ_i be the root and delayradius of Π_i for each $i = 1, ..., k$. Let p_i be the counting path constructed for Π_i and $\tau_i = \tau(G, G_i)$, $1 \leqslant i \leqslant k$. Beginning at time $t_0 = (4\Delta + 4)^2 + 2\Delta + 2$, G sends the number x (as a sequence of signals), digit by digit with the least significant digit first, to G_i via link (G, G_i), for each $i = 1, ..., k$, and releases the circuit pp^{R}. In the following, let i be a fixed index between 1 and k.

The least significant digit of x is received by G_i at time $t_{i,1} = t_0 + \tau_i + 1$. When G_i receives the digit, it forms the circuit $p_i p_i^R$. Let T_i denote the imaginary tape corresponding to circuit $p_i p_i^R$. Starting from time $t_{i,1}$, G_i counts $\theta(p_i)$ and, at the same time, G_i stores the number and the unary numbers $y_{i,1} = 1^{2\Delta_i+2}$ and $z_{i,1} = 1^{2\tau_i+2}$ on T_i. (The unary number $y_{i,1}$ can be obtained since G_i can count $2\Delta_i + 2$ by propagating a signal down and up in Π_i. $z_{i,1}$ can be obtained similarly.) Note that, since $\theta(p_i) \geqslant |y_{i,1}|$, $\theta(p_i) \geqslant |z_{i,1}|$, and $\theta(p_i) \geqslant \Delta + 1 > |x|$, the (unary) numbers x, $y_{i,1}$, and $z_{i,1}$ can be stored on T_i and it takes at most $\theta(p_i)$ steps to store them. Then, starting from time $t_{i,2} = t_{i,1} + \theta(p_i)$, G_i counts $\theta(p_i)^2$ and computes numbers $y_{i,2} = |y_{i,1}|$, $z_{i,2} = |z_{i,1}|$, and $w_i = \theta(p_i)$ (and, of course, stores the numbers on T_i). After this, G_i counts another $3\theta(p_i)^2$ and computes numbers $y_{i,3} = \Delta_i = (y_{i,2} - 2)/2$ and $z_{i,3} = \tau_i = (z_{i,2} - 2)/2$. Starting from time $t_{i,3} = t_{i,2} + 4\theta(p_i)^2$, G_i counts $\theta(p_i)^2$ and computes numbers x^2, $y_{i,3}^2$, and w_i^2. Then G_i counts another $\theta(p_i)^2$ and computes numbers x^3 and $y_{i,3}^3$. Starting from time $t_{i,4} = t_{i,3} + 2\theta(p_i)^2$, G_i counts $\theta(p_i)$ and computes $t_i' = g(\Delta, \Delta_i) - t_{i,4} - \theta(p_i) = a\Delta^3 - a\Delta_i^3 - (4\Delta + 4)^2 - (2\Delta + 2) - (\tau_i + 1) - 6\theta(p_i)^2 - 2\theta(p_i) = ax^3 - ay_{i,3}^3 - 16x^2 - 34x - z_{i,3} - 6w_i^2 - 2w_i - 19$. Thus, number t_i' is stored on T_i by time $t_{i,5} = t_{i,4} + \theta(p_i)$. Then G_i starts to count t_i' at time $t_{i,5}$. When this is done (i.e., at time $t_{i,5} + t_i'$), G_i releases the circuit $p_i p_i^R$ and repeats the above process in subtree Π_i. That is, the synchronization of Π_i begins at time $t_{i,5} + t_i'$. It is easy to verify that $t_{i,5} + t_i' = g(\Delta, \Delta_i)$.

The timing of the operations of G_i is illustrated in Table 1.

TABLE 1

The Timing of the Operations of G_i

Time	Numbers Computed and Stored
$t_{i,1} \to t_{i,1} + \theta(p_i)$	$x = \Delta$ $y_{i,1} = 1^{2\Delta_i+2}$ $z_{i,1} = 1^{2\tau_i+2}$
$t_{i,1} + \theta(p_i) \to t_{i,1} + \theta(p_i)^2 + \theta(p_i)$	$y_{i,2} = \lvert y_{i,1}\rvert = 2\Delta_i + 2$ $z_{i,2} = \lvert z_{i,1}\rvert = 2\tau_i + 2$ $w_i = \theta(p_i)$
$t_{i,1} + \theta(p_i)^2 + \theta(p_i) \to t_{i,1} + 4\theta(p_i)^2 + \theta(p_i)$	$y_{i,3} = (y_{i,2} - 2)/2 = \Delta_i$ $z_{i,3} = (z_{i,2} - 2)/2 = \tau_i$
$t_{i,1} + 4\theta(p_i)^2 + \theta(p_i) \to t_{i,1} + 5\theta(p_i)^2 + \theta(p_i)$	x^2, $y_{i,3}^2$, w_i^2
$t_{i,1} + 6\theta(p_i)^2 + \theta(p_i) \to t_{i,1} + 6\theta(p_i)^2 + \theta(p_i)$	x^3, $y_{i,3}^3$
$t_{i,1} + 6\theta(p_i)^2 + \theta(p_i) \to t_{i,1} + 6\theta(p_i)^2 + 2\theta(p_i)$	$t_i' = ax^3 - ay_{i,3}^3 - 16x^2 - 34x - z_{i,3} - 6w_i^2 - 2w_i - 19$ $= a\Delta^3 - a\Delta_i^3 - (2\Delta + 2) - (4\Delta + 4)^2 - (\tau_i + 1) - 6\theta(p_i)^2 - 2\theta(p_i)$
$t_{i,1} + 6\theta(p_i)^2 + 2\theta(p_i) \to t_{i,1} + 6\theta(p_i)^2 + 2\theta(p) + t_i'$ (Note that $t_{i,1} + 6\theta(p)^2 + 2\theta(p) + t_i' = g(\Delta, \Delta_i)$)	Counting

In order to count t_i', the condition of Lemma 6 must be satisfied, i.e., $t_i' \geqslant 4\theta(p_i)^2 = 4w_i^2$. Since $x > y_{i,3}$, $ax^3 - ay_{i,3}^3 \geqslant ax^2$. According to the construction in Section 3, $x = \Delta \geqslant \tau_i = z_{i,3}$ and $2x + 2 \geqslant w_i$. Since $x > 0$, $x \geqslant \max\{1, (w_i - 2)/2\} \geqslant w_i/4$. Thus, $t_i' = ax^3 - ay_{i,3}^3 - 16x^2 - 34x - z_{i,3} - 6w_i^2 - 2w_i - 19 \geqslant (a-70)x^2 - 8w_i^2 \geqslant (a-70)w_i^2/4 - 8w_i^2 \geqslant (a-102)w_i^2/4$. Let $a = 118$. Then, $t_i' \geqslant (a-102)w_i^2/4 = 16w_i^2/4 = 4w_i^2$.

We have also to ensure that all the numbers that need to be stored on T_i in the above process are of length at most $\theta(p_i) = w_i$. It suffices to make sure that $|x^3|$, $|t_i'|$, and $|w_i^2|$ are at most w_i. Since $w_i \geqslant 2$, $|w_i^2| = \lfloor \log_b w_i^2 \rfloor + 1 \leqslant w_i$ if $b > 2$. We know that $w_i \geqslant x + 1 \geqslant 2$. Since $x^3 \leqslant (w_i - 1)^3$ and $t_i' \leqslant ax^3 \leqslant 118(w_i - 1)^3$, we only need to choose b such that $b > 2$ and $\lfloor \log_b 118(w_i - 1)^3 \rfloor + 1 \leqslant w_i$. Let $b = 11$. Then, it is easy to verify that $\lfloor \log_{11} 118(w_i - 1)^3 \rfloor + 1 \leqslant w_i$.

The above result is summarized in the following theorem.

THEOREM 7. A_{tnud} *is a* $118\Delta^3 + 1$ *time solution of* tfssp-nud.

Now we are ready to give the solution of gfssp-nud.

COROLLARY 8. gfssp-nud *has a* $118\Delta^3 + 2\Delta + 2\tau_{\max} + 5$ *time-bounded solution* A_{gnud}.

Proof. Let N be a given network composed of copies of our intended solution A_{gnud}. As said before, the nodes of N work in two stages. In the first stage, they construct a minimum-delay spanning tree Π of N. In the second stage, they simulate the behavior of A_{tnud} within the tree Π. By Theorem 7, Π (thus, N) will fire in $118\Delta^3 + 1$ steps after the second stage is started, where Δ is the delayradius of N. Thus, it suffices to show that the minimum-delay spanning tree Π can be constructed in $2\Delta + 2\tau_{\max} + 4$ steps, where $\tau_{\max}$ is maximum link delay of N. Note that it is also important for the general of N to know when the construction of Π is done (so it can begin the second stage). We sketch the construction of Π below.

Let G be the general of N. The tree Π can be constructed as follows. To start the construction, G sends the signal *Connect* to all its neighbors. We describe the behavior of an arbitrary node u. Suppose that node u receives the first batch of *Connect*'s at some time t from neighbors $i_1, ..., i_k$, where $1 \leqslant i_1 < \cdots < i_k \leqslant d$. (Note that u must be in the quiescent state *Quiet* at time t.) Then u remembers neighbor i_1 as its parent and, at time $t+1$, sends the signal *Yes* to neighbor i_1, the signal *Complete* to neighbors $i_2, ..., i_k$, and the signal *Connect* to the other neighbors. After time t, if node u receives a *Connect* from some neighbor i, it replies to neighbor i with a *Complete*. If u receives a *Yes* from some neighbor i, it remembers neighbor i as one of its children. Meanwhile, u observes from which neighbor it has received a *Connect* or *Complete*. If u detects that it has received a *Connect*

or *Complete* from each of its neighbors, it sends a *Complete* to the parent. When the general G detects that it has received a *Complete* from each of its neighbors, it knows that the tree Π has been marked. It is easy to see that Π is a minimum-delay spanning tree of N.

Figure 3 shows the minimum-delay spanning tree obtained using the above rules for the network in Fig. 1.

In the above process, each leaf of Π receives the first *Connect* within $\Delta + 1$ steps (after the construction is started) and the last *Connect* or *Complete* within $\Delta + 2\tau_{\max} + 3$ steps. By some straightforward analysis, we can see that G receives the last *Complete* within $2\Delta + 2\tau_{\max} + 4$ steps. ∎

The next corollary is obvious.

COROLLARY 9. A_{tnud} *is a* $118\Phi^3 + 1$ *time-bounded solution of* tfssp-nud *and* A_{gnud} *is a* $118\Phi^3 + 2\Phi + 2\tau_{\max} + 5$ *time-bounded solution of* gfssp-nud.

6. gfssp AND gfssp-nud WITH MORE THAN ONE GENERAL

We can generalize the definition of gfssp (or gfssp-nud) so that a network may have several generals. The generals of a network are independent of each other and may be excited by the external world at different times. (But we still assume that a general can receive an external excitation only when it is quiescent.) The problem is to construct an fa A such that, for any network N composed of A's, if one or more generals of N are excited, then N fires at some time. Let gfssp(k) (gfssp(∞)) denote the extension of gfssp to networks with at most k (arbitrary number of respectively) generals. The problems gfssp-nud(k), gfssp-nud(∞), tfssp(∞), and tfssp-nud(∞) are defined similarly.

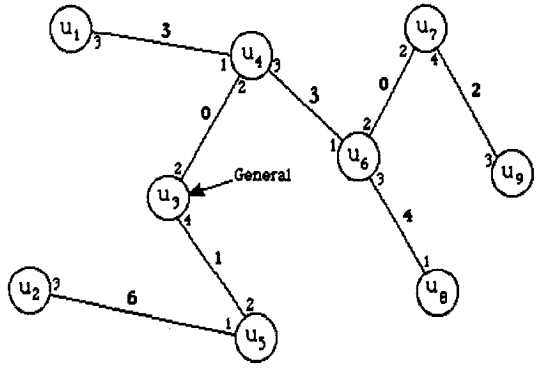

FIG. 3. The minimum-delay spanning tree for the network in Fig. 1.

A natural thought on synchronizing a network with multiple generals is to suppress all but one (called *leader*) excited generals so that a solution of gfssp (or gfssp-nud) can be simulated. The difficulty with this approach is that, since the nodes are all identical fa's and the network may be symmetric, it is hard to break the tie among the excited generals and elect a leader. In fact, sometimes it is impossible to break the tie. Interestingly, we can show that for each fixed k, gfssp(k) and gfssp-nud(k) have solutions.

Some more definitions are necessary. Let N be a network and Π be a tree-structured subnetwork of N. Let u_1 be a node (of N) belonging to Π and $e=(u_1, u_2)$ be a link (of N) incident on u_1. If link e does not belong to Π, then e is called a *side link* of u_1 and u_2 is called a *relative* of u_1. Note that a relative of u_1 may or may not belong to Π. A side link (relative) of Π is a side link (relative) of any node belonging to Π. Two disjoint tree-structured subnetworks Π_1 and Π_2 of N are said to be *adjacent* if some nodes belonging to Π_1 are relatives of Π_2 and vice versa.

We say that a solution A (of some problem) is $h(D)$ (or $h(\Phi, \tau_{\max})$) time-bounded for some function h if, for any appropriate network N composed of A's with diameter D (or delaydiameter Φ and maximum link delay $\tau_{\max}$), N fires in at most $h(D)$ (or $h(\Phi, \tau_{\max})$) steps after the first general being excited. Note that the parameters radius and delayradius are not defined for networks with multiple generals.

THEOREM 10. (1) *For each* $k \geq 2$, gfssp(k) *has an* $O(D)$ *time-bounded solution.*

(2) *For each* $k \geq 2$, gfssp-nud(k) *has an* $O((\Phi+\tau_{\max})^3)$ *time-bounded solution.*

Proof. We give only the proof of (2). (1) can beproven similarly. The only change that needs to be made is that in the proof of (2), we use the solution A_{gnud} of gfssp-nud, whereas in the proof of (1), the $4R$ time-bounded solution of gfssp given by Nishitani and Honda should be used.

Let k be any integer greater than or eqal to 2. Let $A_{\text{gnud}(k)}$ denote our intended solution of gfssp-nud(k). We first give the idea behind the construction of $A_{\text{gnud}(k)}$.

Let N be a network composed of $A_{\text{gnud}(k)}$'s with at most k generals. The nodes of N will go through a sequence of simulations of A_{gnud}'s. After a general is excited, it initiates a simulation of A_{gnud}'s in N. According to the construction of A_{gnud}, an excited general first tries to construct a minimum-delay spanning tree of N as described in the proof of Corollary 8. But because of the possible existence of more than one excited general, the tree constructed by each general will most likely not be a minimum-delay

spanning tree of N. Generally, each general obtains a tree-structured subnetwork of N. Suppose that k' generals $G_1, ..., G_{k'}$ are excited eventually, $k' \leqslant k$. (Note that, since the general G_i can receive an external excitation only when it is qiescent, G_i must be excited before it gets involved in the tree-construction processes initiated by other generals, i.e., its state must be set to *Start* before it receives a signal *Connect* from some neighbor.) Let $\Pi_1, ..., \Pi_{k'}$ be the trees constructed by generals $G_1, ..., G_{k'}$, respectively. It is easy to see that the trees $\Pi_1, ..., \Pi_{k'}$ form a partition of N.

The simulation of A_{gnud}'s is carried out independently in the trees $\Pi_1, ..., \Pi_{k'}$. When the nodes of some tree detects that the simulated A_{gnud}'s are about to enter the state *Fire*, they enter a special state (call it *Test*) instead. Then the nodes check if their relatives entered *Test* before, at the same time as, or after they did. If there exist adjacent trees Π_i and Π_j whose nodes entered *Test* at different times, then the tie between G_i and G_j can be broken as follows: if the nodes of Π_i (Π_j) entered *Test* earlier, then G_i (G_j) is kept as a general and G_j (G_i) is degraded to a soldier. In this case, at least one general will be degraded. The undegraded generals will then initiate more simulations of A_{gnud}'s in N to degrade more generals. Now suppose that all the nodes of N entered *Test* simultaneously. Each general G_i ($1 \leqslant i \leqslant k'$) will initiate a new simulation of A_{gnud}'s, but only within the scope of tree Π_i. Thus, the simulation of A_{gnud}'s is again carried out independently in trees $\Pi_1, ..., \Pi_{k'}$. As before, the nodes will enter the state *Test* and do the above-mentioned tests. If there are adjacent trees whose nodes entered *Test* at different times, then some generals will be degraded as described above. On the other hand, if all the nodes of N again entered *Test* at the same time, then there is no way to decide which of $G_1, ..., G_{k'}$ should be degraded. But, if this happens, then it is easy to see that if we let each G_i ($1 \leqslant i \leqslant k'$) initiate another simulation of A_{gnud}'s in Π_i independently, then all the nodes of N will again enter *Test* simultaneously. This gives us a way to synchronize N without breaking the tie among $G_1, ..., G_{k'}$.

Let the undegraded generals repeat the above process (i.e., simulation of A_{gnud}'s, test, and degradation) for $2k-2$ times. Then it can be shown that if we let each of the resulting undegraded generals initiate a simulation of A_{gnud}'s within the tree that is currently in, then all the nodes of N will enter state *Test* at the same time.

Now we give the details of $A_{\text{gnud}(k)}$. In the following, $A^1_{\text{gnud}}, ..., A^{2k-1}_{\text{gnud}}$ denote $2k-1$ copies of the fa A_{tnud}. To make these fa's distinct, the states and signals of A^i_{tnud} are marked by superscript i, $1 \leqslant i \leqslant 2k-1$. Thus, the fa A^i_{tnud} has states $Quiet^i$, $Start^i$, $Fire^i$, etc. and signals $\$^i$, $Connect^i$, Yes^i, $Complete^i$, etc., $1 \leqslant i \leqslant 2k-1$. For convenience, let $Quiet^1 = Quiet$, $Start^1 = Start$, and $Fire^{2k-1} = Fire$. The states $Fire^1, ..., Fire^{2k-2}$ will serve as the special testing state.

Let the network N be as in the above and Φ and $\tau_{\max}$ be the delay-diameter and maximum link delay of N respectively. For convenience, we assume that each node of N has, besides its finite-state control, a d-bit mask register. The use of these mark registers is to make some links of N conceptually nonexistent to a simulation of A^i_{gnud}'s ($1 \leqslant i \leqslant 2k-1$). (This is necessary when we desire a simulation to be performed within a specific subnetwork of N.) Each bit of a node's mask register corresponds to a link incident on the node. A node can *mask* (or *unmask*) a link incident on it by setting the corresponding bit to 1 (or 0). The bits of a mask register are initially set to 0's. Conventionally, if a node u_1 masks some link $e=(u_1, u_2)$, then u_1 will not consider u_2 as a neighbor in the subsequent simulations of A^i_{gnud}'s, unless it unmasks e again. A link $e=(u_1, u_2)$ is said to be *masked* if both u_1 and u_2 have masked e. Clearly, if a links is masked, then it is nonexistent to the present simulation of A^i_{gnud}'s.

We describe how the nodes of N operate. The nodes first simulate A^1_{gnud}'s. Suppose that the generals $G_1, ..., G_{k'}$ are excited eventually, $k' \leqslant k$. Let $\Pi_1, ..., \Pi_{k'}$ be the trees constructed by $G_1, ..., G_{k'}$, respectively. Recall that $\Pi_1, ..., \Pi_{k'}$ form a partition of N. Moreover, tree Π_i is rooted at G_i and has delayradius $\Delta_i \leqslant \Phi$, $1 \leqslant i \leqslant k'$. After a node (of some tree) enters the state $Fire^1$, it tests for each of its relatives, whether the relative entered state $Fire^1$ before, at the same time as, or after it did. This can be done as follows.

Let u_1 and u_2 be relatives of each other and τ be the delay of link (u_1, u_2). (Note that u_1 and u_2 may be in two different trees.) Suppose that u_1 and u_2 enter state $Fire^1$ at times t_1 and t_2, respectively. It is very easy for u_1 and u_2 to know if $t_1 \geqslant t_2 + \tau$ or $t_2 \geqslant t_1 + \tau$. Thus, assume $|t_1 - t_2| < \tau$. After entering $Fire^1$, each of u_1 and u_2 sends the other party two signals, *Head* and *Tail*, in two consecutive steps. When u_1 receives a *Head*, it sends the signal back to u_2 immediately. When u_1 receives a *Tail* that originated from u_2, it sends the signal back immediately. When u_1 receives a *Tail* that originated from itself, it keeps the signal for one step and then sends the signal back. u_2 works in an analogous way. u_1 and u_2 can keep track of the origin of these *Head*'s and *Tail*'s by counting the parity of their visits. For example, when u_1 receives a *Head* (or *Tail*) for the $2i$th time ($i \geqslant 1$), it knows that the signal originated from itself. Let u be either u_1 or u_2. The node u sends back u_1's *Head* and u_2's *Head* every $2\tau + 2$ steps and sends back u_1's *Tail* and u_2's *Tail* every $2\tau + 3$ steps. Therefore, if we observe for a sufficiently long time, we are certain to see that u sends u_1's *Head* immediately after sending u_2's *Tail* and also see that u sends u_2's *Head* immediately after sending u_1's *Tail*. What is important for deciding whether $t_1 < t_2$, $t_1 = t_2$, or $t_1 > t_2$ is the relative timing of these two events. Let T_0 be the first time such that u sends u_2's *Tail* at time T_0 and u sends u_1's *Head* at time $T_0 + 1$. Let T_1 be the first time such that $T_1 \geqslant T_0$ and u sends

u_1's *Tail* at time T_1 and T_2 be the first time such that $T_2 \geqslant T_0$ and u sends u_2's *Head* at time T_2. Then, we have

(1) $t_1 < t_2$ iff $T_2 > T_1 + 1$

(2) $t_1 = t_2$ iff $T_2 = T_1 + 1$

(3) $t_1 > t_2$ iff $T_2 < T_1 + 1$

Using the above property, u can know if $t_1 < t_2$, $t_1 = t_2$, or $t_1 > t_2$. It is easy to see that the above test takes at most $\tau(2\tau + 2)$ steps.

Let i be any index between 1 and k'. Consider the tree Π_i. Suppose that the nodes of Π_i enter state $Fire^1$ at some time t. After doing the above-mentioned test, each node of Π_i masks all its side links leading to relatives that entered state $Fire^1$ at time t. Meanwhile, the nodes report the test results to G_i as follows. When a leaf detects that none of its neighbors entered $Fire^1$ before time t, it sends a signal *Continue* to its parent. When an internal node detects that none of its neighbors entered $Fire^1$ before time t and each child has sent a *Continue*, it sends a *Continue* to its parent. If the root G_i detects that none of its neighbors entered $Fire^1$ before time t and each child has sent a *Continue*, it knows that no relatives of Π_i entered state $Fire^1$ before time t. Clearly, if no relatives of Π_i entered state $Fire^1$ before time t, G_i should detect the fact by the time $t + \tau_{\max}(2\tau_{\max} + 2) + \Delta_i + 1$.

If no relatives of Π_i entered state $Fire^1$ before time t, then G knows that it should not be degraded at the moment. Thus, it assumes the position of an excited A^2_{gnud} and initiate a simulation of A^2_{gnud}'s in N. Note that, since some links of N may have been masked, the simulation will actually be performed within a subnetwork of N. In the extreme, if all relatives of Π_i entered state $Fire^1$ at time t, then all side links of Π_i have been masked and Π_i is "disconnected" from the rest of N. In this case, the simulation will be performed within Π_i.

When a node is involved in the simulation of A^2_{gnud}'s, the node is forced to abandon its current work and begin acting like an A^2_{gnud}. Thus, if a node is in some state not belonging to A^2_{gnud} and receives a $Connect^2$ from some neighbor, it automatically assumes the position of a soldier and begins acting like an A^2_{gnud}. Of course, the node has to treat the special signals such as *Head* and *Tail* properly. If the node receives a *Head* or *Tail*, it should send back proper acknowledgments.

Note that, if some of the relatives of the tree Π_i entered $Fire^1$ before time t, then G_i will never initiate a simulation of A^2_{gnud}'s. Thus, G_i will assume the position of a soldier when it is involved in a simulation of A^2_{gnud}'s initiated by some other general, i.e., it will be degraded automatically.

The above process will be repeated for a total number of $2k - 1$ times by the undegraded generals. Generally, in the ith round $(2 \leqslant i \leqslant 2k - 2)$, the

nodes simulate A^i_{gnud}'s. When a node enters the state $Fire^i$, it tests whether its relatives entered state $Fire^i$ before, at the same time as, or after it did. Note that, the presently masked links are inexistent only to the simulation of A^i_{gnud}'s. Thus, the above test is done for all relatives of the node in N. The node then masks all its side links leading to relatives that entered $Fire^i$ at the same time as it did and unmasks its others side links. Also based on such tests, a general decides if it should initiate a simulation of A^{i+1}_{gnud}'s. The last (i.e., the $(2k-1)$st) round ends with the nodes entering the state $Fire^{2k-1} = Fire$.

Correctness and Time Complexity. We first prove that all nodes of N enter the state $Fire = Fire^{2k-1}$ exactly at a same time; i.e., N eventually fires. Let S_i denote the set of the undegraded generals in the ith round, $1 \leqslant i \leqslant 2k-1$. (Note that the generals may enter a same round at different times.) Clearly, $\varnothing \neq S_{2k-1} \subseteq \cdots \subseteq S_1 = \{G_1, ..., G_{k'}\}$. If $|S_{2k-1}| = 1$, then obviously all nodes of N enter $Fire$ at a same time. Thus, suppose $|S_{2k-1}| > 1$. Observe that, according to the above construction, the trees constructed by the members of S_i in the ith round form a partition of N, for any $1 \leqslant i \leqslant 2k-1$.

Since $k' \leqslant k$, it is easy to see that there must exist some r, $1 \leqslant r \leqslant 2k-3$, such that $S_r = S_{r+1} = S_{r+2}$. Let $S_r = \{G'_1, ..., G'_s\}$ and $\Pi'_1, ..., \Pi'_s$ be the trees constructed by $G'_1, ..., G'_s$ respectively in the rth round. Since $S_r = S_{r+1}$ and $\{\Pi'_1, ..., \Pi'_s\}$ is a partition of N, it is easy to see that all nodes of N must enter the state $Fire^r$ at the same time t. Suppose that it takes (exactly) $t_{1,i}$ steps for G'_i to detect the fact that all relatives of Π'_i entered state $Fire^r$ also at time t, $1 \leqslant i \leqslant s$. By time $t + t_{1,i}$, all the side links of Π'_i are marked, i.e., Π'_i is totally "disconnected" from the rest of N, $1 \leqslant i \leqslant s$. Suppose that it takes (again, exactly) $t_{2,i}$ steps for G'_i to make the nodes of Π'_i enter the state $Fire^{r+1}$ (by initiating a simulation of A^{r+1}_{gnud}'s), $1 \leqslant i \leqslant s$. Since $S_{r+1} = S_{r+2}$, all nodes of N must enter $Fire^{r+1}$ simultaneously at a time $t + \delta$ for sme δ. Thus, $t_{1,1} + t_{2,1} = \cdots = t_{1,s} + t_{2,s} = \delta$. So, at time $t + \delta + t_{1,i}$, G'_i detects the fact that all relatives of Π'_i entered $Fire^{r+1}$ at time $t + \delta$, masks all the side links of Π'_i, and initiates a simulation of A^{r+2}_{gnud}'s in tree Π'_i, $1 \leqslant i \leqslant s$. Hence, the nodes of Π'_i enter the state $Fire^{r+2}$ at time $t + \delta + t_{1,i} + t_{2,i} = t + 2\delta$, $1 \leqslant i \leqslant s$. That is, all nodes of N enter the state $Fire^{r+2}$ simultaneously at time $t + 2\delta$. If $r + 2 = 2k - 1$, then we are done. Otherwise, we have $S_{r+3} = S_{r+2}$ and the same argument can be used to show that all nodes of N enter the state $Fire^{r+3}$ at a same time. By repeating the argument for certain number of times, we have that $S_{2k-1} = \cdots = S_r$ and all nodes of N enter the state $Fire^{2k-1}$ at a same time.

We now prove that N fires in $c((\Phi + \tau_{\max})^3)$ steps, for some constant c depending only on k. Suppose that N fires at time t_f. For each i

$(1 \leqslant i \leqslant 2k-1)$ and $G \in S_i$, let $t_{G,i}$ denote the time when G enters the ith round, $\Pi_{G,i}$ denote the tree constructed by G in the ith round (note that $\Pi_{G,i}$ is rooted at G), and $\Delta_{G,i}$ denote the delayradius of $\Pi_{G,i}$. Then $t_{G,i+1} - t_{G,i} \leqslant (118\Delta_{G,i}^3 + 2\Delta_{G,i} + 2\tau_{\max} + 5) + \tau_{\max}(2\tau_{\max} + 2) + (\Delta_{G,i} + 1) \leqslant c_1(\Delta_{G,i} + \tau_{\max})^3$ for some constant c_1, $1 \leqslant i \leqslant 2k-2$ and $G \in S_{i+1}$, and $t_f - t_{G,2k-1} \leqslant 118\Delta_{G,2k-1}^3 + 2\Delta_{G,2k-1} + 2\tau_{\max} + 5 < c_1(\Delta_{G,2k-1} + \tau_{\max})^3$, $G \in S_{2k-1}$. Thus, it suffices to show that $\Delta_{G,i} \leqslant c_2(\Phi + \tau_{\max})$ for some constant c_2, $1 \leqslant i \leqslant 2k-1$ and $G \in S_i$. We prove by induction on i that $\Delta_{G,i} \leqslant (5k)^{i-1}(\Phi + \tau_{\max})$, $1 \leqslant i \leqslant 2k-1$ and $G \in S_i$.

As mentioned before, $\Delta_{G,1} \leqslant \Phi$, for all $G \in S_1$. Suppose that $\Delta_{G,i} \leqslant (5k)^{i-1}(\Phi + \tau_{\max})$ for all $G \in S_i$. Let G be any member of S_{i+1}. We show that $\Delta_{G,i+1} \leqslant (5k)^i(\Phi + \tau_{\max})$. Let u be any node of tree $\Pi_{G,i+1}$ and τ denote the delay between G and u in tree $\Pi_{G,i+1}$. Since $\{\Pi_{X,i} | X \in S_i\}$ is a partition of N, u must be in a unique tree $\Pi_{G',i}$ for some $G' \in S_i$. If $G' = G$, then obviously $\tau \leqslant$ the delay between G and u in $\Pi_{G,i} \leqslant (5k)^{i-1}(\Phi + \tau_{\max})$. Otherwise, there must exist $H_1, ..., H_r \in S_i$, such that $\Pi_{G,i}$ is adjacent to $\Pi_{H_1,i}$, $\Pi_{H_j,i}$ is adjacent to $\Pi_{H_{j+1},i}$, $1 \leqslant j < r$, and $\Pi_{H_r,i}$ is adjacent to $\Pi_{G',i}$. Then it is easy to see that $\tau \leqslant \Delta_{G,i} + 2 + \tau_{\max} + \sum_{j=1}^{r}(2\Delta_{H_j,i} + \tau_{\max} + 3) + \Delta_{G',i} \leqslant 2(r+1)(5k)^{i-1}(\Phi + \tau_{\max}) + (r+1)\tau_{\max} + 3r + 2$. Since $\Phi \geqslant 1$ and $t \leqslant k' - 2 \leqslant k - 2$, $\tau \leqslant 2(r+1)(5k)^{i-1}(\Phi + \tau_{\max}) + (r+1)\tau_{\max} + 3(r+1)\Phi \leqslant 2(r+1)(5k)^{i-1}(\Phi + \tau_{\max}) + 3(r+1)(\Phi + \tau_{\max}) \leqslant 5(r+1)(5k)^{i-1}(\Phi + \tau_{\max}) \leqslant (5k)^i(\Phi + \tau_{\max})$. Thus, $\Delta_{G,i+1} \leqslant (5k)^i(\Phi + \tau_{\max})$.

Hence, $A_{\text{gnud}(k)}$ is an $O((\Phi + \tau_{\max})^3)$ time-bounded solution of gfssp-nud(k). ∎

On the other hand, it is not hard to show that when there are unbounded number of generals, the synchronization of an arbitrarily connected network is not achievable.

THEOREM 11. *There are no solutions for* gfssp(∞) *or* gfssp-nud(∞).

Proof. We prove the theorem for the case $d = 2$. The generalization of the proof to $d > 2$ is trivial. Since gfssp(∞) is a special case of gfssp-nud(∞), it suffices to prove that there is no solution for gfssp(∞).

Suppose that gfssp(∞) has a solution $A_{g(\infty)}$. For each $i \geqslant 3$, let R_i denote the ring-structured network (ring for short) defined as follows. The ring R_i consists of i nodes $u_1, ..., u_i$ and i links $(u_1, u_2), ..., (u_{i-1}, u_i), (u_i, u_1)$. Each u_j, $1 \leqslant j \leqslant i$, represents a copy of $A_{g(\infty)}$ and is a general of R_i. The links all have 0 transmission delays. For each $j = 1, ..., i$, the link $(u_j, u_{j+1 \bmod i})$ is connected to the 2nd terminal of u_j and the 1st terminal of $u_{j+1 \bmod i}$. Figure 4 illustrates the ring R_6.

First, let us consider the ring R_3. Suppose that all three nodes of R_3 are simultaneously excited at time 0. Since $A_{g(\infty)}$ is a solution of gfssp(∞), R_3 fires at some time t_0. Clearly, $t_0 \geqslant 2$. It is easy to see that behaviors of the

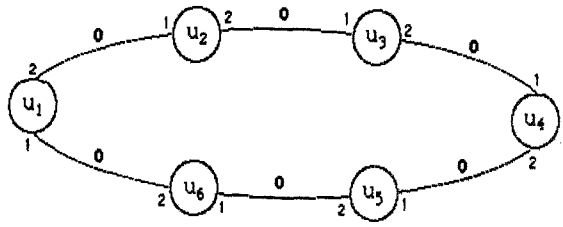

FIG. 4. The ring R_6.

three nodes (of R_3) are exactly the same during the synchronization process, i.e., the nodes always enter the same state and send/receive the same signals. Let q_t $(o_{1,t}, o_{2,t})$ denote the state (the 1st and 2nd output signals, respectively) of the nodes at time t, $1 \leqslant t \leqslant t_0$. Note that $q_{t_0} = \textit{Fire}$.

Now consider the ring R_{4t_0-2}. Suppose that the nodes $u_1, ..., u_{2t_0-1}$ are simultaneously excited at time 0 and the nodes $u_{2t_0}, ..., u_{4t_0-2}$ are never excited. Let t be any integer such that $1 \leqslant t \leqslant t_0$. It is easy to see that (1) for any i satisfying $t \leqslant i \leqslant 2t_0 - t$, the state and the 1st and 2nd output signals of the node u_i at time t are q_t, $o_{1,t}$, and $o_{2,t}$, respectively; (2) for any i satisfying $2t_0 + t - 1 \leqslant i \leqslant 4t_0 - t - 1$, the state and the 1st and 2nd output signals of the node u_i at time t are *Quiet*, \$, and \$ respectively. Thus, the node u_{t_0} enters state $q_{t_0} = \textit{Fire}$ at time t_0 while the node u_{3t_0-1} remains in state *Quiet*. This contradicts the assumption that $A_{g(\infty)}$ is a solution of gfssp(∞). Hence, gfssp(∞) does not have a solution. ∎

This result is in great contrast with the following theorem which states that, no matter how many generals may exist, it is always possible to synchronize a tree.

THEOREM 12. (1) tfssp(∞) *has an* $O(D)$ *time-bounded solution.*

(2) tfssp-nud(∞) *has an* $O(\Phi^3)$ *time-bounded solution.*

Proof. We prove only (2). The proof of (1) is similar.

We sketch a solution of tfssp-nud(∞). Let Π be an *unrooted* tree with an arbitrary number of generals. Each node of Π is a copy of our intended solution of tfssp-nud(∞). Let Φ and $\tau_{\max}$ be the delaydiameter and maximum link delay of Π, respectively. For simplicity, assume $\Phi > 0$, i.e., Π is not a single node. Suppose that the first general is excited at time 0. Clearly, $\tau_{\max} \leqslant \Phi$.

The basic idea is to make one or two nodes distinguished from others so that the problem can be reduced to gfssp-nud(2). The nodes work as follows. After an internal general (i.e., a general who is an internal node) is excited, it sends the signal *Spread* to all its neighbors. If an excited

general is a leaf, it sends the signal *Focus* to its only neighbor. When an internal node receives some *Spread*'s, it spreads the signal *Spread* to all its neighbors. When a leaf receives a *Spread* from its neighbor, it replies to the neighbor with a signal *Focus*. Meanwhile, each node remembers the indices of the neighbors from whom it has received the signal *Focus*. If a node detects that it has received the signal *Focus* from all but one of its neighbors, it sends the signal *Focus* to the neighbor from whom it has not yet received a *Focus*. If a node detects that it has received a *Focus* from all its neighbors, it assumes the position of an excited $A_{\text{gnud(2)}}$ and initiates a simulation of $A_{\text{gnud(2)}}$'s in Π. (Recall that $A_{\text{gnud(2)}}$ is an $O((\Phi+\tau_{\max})^3)$ time-bounded solution of gfssp-nud(2).) A node of Π will enter state *Fire* when the simulated $A_{\text{gnud(2)}}$ enters *Fire*.

The following are easy to show: (1) at least one node must assume the position of an excited $A_{\text{gnud(2)}}$ by time $2\Phi+2$ and (2) at most two nodes may assume such a position. Thus, the simulation of $A_{\text{gnud(2)}}$'s will make Π fire by the time $c(\Phi+\tau_{\max})^3$, where c is some constant independent of Π. Since $\tau_{\max} \leqslant \Phi$, Π fires no later than time $8c\Phi^3$.

Hence, tfssp-nud(∞) has an $O(\Phi^3)$ time-bounded solution. ∎

7. Concluding Remarks

It would be interesting to know if the time complexity in Theorem 7 can be improved to, e.g., $O(\Delta^2)$. Note that, since the best-known solution of fssp-ud (given by Varshavsky *et al.*) has firing time $\Omega(\Delta\tau)$, where τ is the delay of a link, it seems unlikely that an $O(\Delta)$ time-bounded solution of tfssp-nud exists.

Acknowledgment

The author thanks the referees for carefully reading an earlier version of the paper. Their detailed comments and suggestions have helped improve the presentation of the paper.

Received September 22, 1989; final manuscript received August 22, 1990

References

Balzer, R. (1967), An 8-state minial time solution to the firing squad synchronization problem, *Inform. Control* **10**, 22.

Culik, K. (1989), Variations of the firing squad problem and applications, *Inform. Process. Lett.* **30**, 153.

Grasselli, A. (1975), Synchronization of cellular arrays: The firing squad problem in two dimensions, *Inform. Control* **28**, 113.

KOBAYASHI, K. (1977), The firing squad synchronization problem for two dimensional arrays, *Inform. Control* **34**, 177.

KOBAYASHI, K. (1978a), The firing squad synchronization problem for a class of polyautomata networks, *J. Comput. System Sci.* **17**, 300–318.

KOBAYASHI, K. (1978b), On the minimal firing time of the firing squad synchronization problem for polyautomata networks, *Theoret. Comput. Sci.* **7**, 149.

MOORE, E. (1959), The shortest path through a maze, *in* "Proceedings of International Symposium in the Theory of Switching, 1959," pp. 285–292.

MOORE, E. (1964), "Sequential Machines, Selected Papers," Addison–Wesley, Reading, MA.

MINSKY, M. (1967), "Finite and Infinite Machines," Prentice–Hall, Englewood Cliffs, NJ.

MAZOYER, J. (1986), An overview of the firing squad synchronization problem, *in* "Automata Networks" (C. Choffrut, Ed.), pp. 82–93, Lecture Notes in Computer Science, Vol. 316, Springer-Verlag, Berlin/New York.

MOORE, F., AND LANGDON, G. (1968), A generalized firing squad problem, *Inform. Control* **12**, 212.

NISHITANI, Y., AND HONDA, N. (1977), The firing squad synchronization problem for graphs, *in* "Studies on Polyautomata" (1976 Research Group on Polyautomata, their Structures and Functions), pp. 158–188.

ROMANI, F. (1976), Cellular automata synchronization, *Inform. Sci.* **10**, 299.

ROMANI, F. (1978), The parallelism principle: Speeding up the cellular automata synchronization, *Inform. Control* **36**, 245.

ROSENSTIEHL, P., FISKEL, J., AND HOLLIGER, A. (1973), Intelligent graphs: Networks for finite automata capable of solving graph problems, *in* "Graph Theory and Computing" (R. Read, Ed.), Academic Press, New York.

SHINAHR, I. (1974), Two and three dimensional firing squad synchronization problems, *Inform. Control* **24**, 163.

SZWERINSKI, H. (1982), Time optimal solution of the firing squad synchronization problem for n-dimensional rectangles with the general at an arbitrary position, *Theoret. Comput. Sci.* **19**, 305–320.

VARSHAVSKY, V., MARAKHOVSKY, V., AND PESCHANSKY, V. (1970), Synchronization of interacting automata, *Math. System Theory* **14**, 212.

WAKSMAN, A. An optimal solution to the firing squad synchronization problem, *Inform. Control* **9**, 66–78.